МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсивная обработка иерархических списков

Студент гр. 8304 ___ Птухов Д. А.

Преподаватель ___ Фиалковский М. С.

Санкт-Петербург
2019

Цель работы.

Получить опыт работы с иерархическим списком и его рекурсивной обработкой (Вариант 24).

Постановка задачи.

1) Проанализировав условие задачи, разработать эффективный алгоритм для считывания данных и их обработки.
2) сопоставить рекурсивное решение с итеративным решением задачи;
3) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

Описание алгоритма.

Для начала необходимо эффективно считать данные из входного потока или входного файла. Для хранения был реализован иерархический список на основе класса Node содержащей в себе два std::variant поля. Первое – отвечает за кол-во аргументов для хранящейся операции (2 – для бинарных, 1 - унарных), второе – отвечает за хранимое значение в текущем подсписке (char – для операции, int – для значения атома). Также в классе Node содержится метод evaluate(), который предназначен для прохода по ранее созданному списку. Для считывания данных была написана функция рекурсивная ListFormation. При очередном вызове функции ListFormation ведется поиск операции, содержащейся в рассматриваемом блоке, далее идет формирование 2-х аргументов для бинарных операций и 1-ого для унарных операций. После формирования элементов (элемента) их (его) значения записываются в новые объекты класса Node и они становятся элементами исходного списка содержащего ранее найденную операцию.

Спецификация программы.

Программа предназначена для вычисления выражения введенного пользователем.

Программа написана на языке C++ входными данными является либо строка с выражением и строка со значениями переменных, либо путь до файла с вышеописанными строками.

Описание функций.

1) RemakePowerToInt

Объявление функции:

```
bool Arithmetic::RemakePowerToInt()
```

Данная функция является методом класса Arithmetic содержащегося поле data_ отвечающее за хранение выражения и поля dict сопоставляющего именам переменных их значения. Она замена каждую подстроку вида “power(a, b)” на результат число равное a^b. Данная замена осуществляется при помощи регулярного выражения (определяющего корректность записи функции power) и последовательным получением значений слева и справа от запятой. Далее при помощи функции pow вычисляется значение a^b и вставляется на место ранее удаленной power(a, b).

2) ListFormation

Объявление функции.

```
bool Arithmetic::ListFormation(Node* nd, const std::string s, std::ostream& out)
```

Данная функция осуществляет запись входных данных в иерархический список при помощи определения операции, содержащейся в рассматриваемом блоке и ее аргументов. Выходной поток используется для вывода отладочной информации.

3) ExtractVariableValues

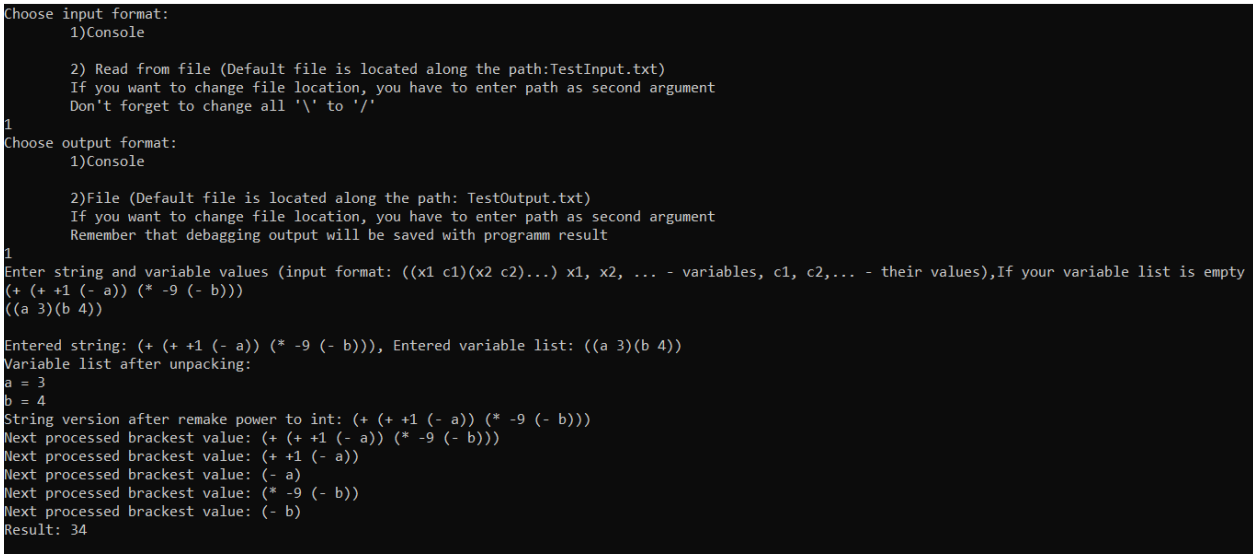
Объявление функции:

```
bool Arithmetic::ExtractVariableValues(std::string value, std::ostream& out)
```

Данная функция осуществляет проверку строки содержащей информацию о переменных. Если строка прошла проверку, то осуществляется сопоставление именам переменных их значений и занесение данной информации в словарь.

Тестирование.

Пример вывода программы:



Входные данные	Выходные данные
(+ a (+ b c)) ((a +0)(b -0)(c -10))	Result: -10
(* (+ 3 (* a b)) (- 3 (* (- a) (- c)))) ((a 1)(b 2))	Incorrect expression!
(* (+ 3 (* a b)) (- 3 (* (- a) (-))))	Incorrect expression!
gtewiuBGEWOugbO'IGENo'gn (888888888888888888888)	Incorrect variable list!
(+ (+ Yo Yo) (+ Yo (+ Yo Yo))) ((Yo 100))	Result: 500

(* ccc (+ (+ aaa power(aaa, bbb)) bbb)) ((aaa -1)(bbb 2)(ccc 3))	Result: 6
(+ a b	Oops, you forgot about close bracket(s)

Выводы.

В ходе работы был получен опыт работы с иерархическими списками и их рекурсивной обработкой, а также с классами и структурами данных.

Приложение.

```
#include "arithmetic.h"

//Выбор входного потока
std::string InputDialog();

//Выбор выходного потока
std::string OutputDialog();

//функция создает путь или если он не был передан заменяет его дефолтным
std::string MakePath(std::string s, std::string const default_path);

//вызов функций обработки строки
void CallAndCheck(StringPair data_, std::ostream& out);

//спуск вниз по списку или получение значения текущего атома
int Node::evaluate()
{
   if (std::holds_alternative<char>(value))
   {
         if (std::get<char>(value) == '+')
               return std::get<NodePair>(arguments).first->evaluate() +
std::get<NodePair>(arguments).second->evaluate();
         if (std::get<char>(value) == '*')
               return std::get<NodePair>(arguments).first->evaluate() *
std::get<NodePair>(arguments).second->evaluate();
         if (std::get<char>(value) == '-')
         {
               if (std::holds_alternative<NodePair>(arguments))
                     return std::get<NodePair>(arguments).first->evaluate()
- std::get<NodePair>(arguments).second->evaluate();
               else
                     return (-1) * std::get<Node*>(arguments)->evaluate();
         }
   }
   else
         return std::get<int>(value);
}

//изменение значения хранящегося в классе arithmetic
void Arithmetic::SetStringValue(std::string const new_data_)
{
   data_ = new_data_;
}

//получение значения хранящегося в классе arithmetic
std::string Arithmetic::GetStringValue()
{
   return data_;
}

//проверка расстановки скобок
int Arithmetic::CheckBrackets()
{
   //ocnt - open cnt, ccnt - close cnt
   int ocnt = 0, ccnt = 0;
```

```
    for (char i : data_)
    {
        if (i == '(')
            ocnt++;

        if (i == ')')
            ccnt++;
    }

    return ocnt - ccnt;
}

//добавление новой пары ключ/значение в словарь
void Arithmetic::UpdateMap(const std::string key, int value)
{
    dict[key] = value;
}

//проверка на нахождение переданного аргумента в словаре
bool Arithmetic::InMap(const std::string value)
{
    return dict.find(value) != dict.end();
}

//Замена строки вида power(a, b) на значение данной операции
bool Arithmetic::RemakePowerToInt()
{
    size_t PowerStartInd = data_.rfind("power(");
    if (PowerStartInd == std::string::npos)
        return 1;

    size_t tmp = PowerStartInd;
    while (data_[++tmp] != ')') {}

    size_t PowerEndInd = ++tmp;
    std::string PowerData(data_.begin() + PowerStartInd, data_.begin() +
PowerEndInd);
    std::regex regular("power\\([a-zA-Z0-9_]+\\s*,\\s*[a-zA-Z0-9_]+\\)");

    if (!std::regex_match(PowerData, regular))
        return 0;

    std::string VariableName = "";
    tmp = 6;

    while (PowerData[tmp] != ' ' && PowerData[tmp] != ',') { VariableName
+= PowerData[tmp]; tmp++; }
    IntBoolPair value1 = PowerHelper(VariableName);
    if (!value1.second)
        return 0;

    VariableName.clear();
    tmp = PowerData.find(',');

    while (PowerData[++tmp] != ')') { if (PowerData[tmp] != ' ')
VariableName += PowerData[tmp]; }
    IntBoolPair value2 = PowerHelper(VariableName);
```

```
    if (!value2.second)
        return 0;

    std::string s = std::to_string(static_cast<int>(std::pow(value1.first,
value2.first)));

    data_.erase(data_.begin() + PowerStartInd, data_.begin() +
PowerEndInd);
    data_.insert(data_.begin() + PowerStartInd, s.begin(), s.end());

    return RemakePowerToInt();

}

//помошник предыдущей функции
IntBoolPair Arithmetic::PowerHelper(std::string VariableName)
{
    int VariableValue;

    if (!InMap(VariableName))
    {
        std::istringstream stream(VariableName);
        stream >> VariableValue;
        if (stream.fail() || stream.peek() != EOF)
            return std::make_pair(0, 0);
    }
    else
        VariableValue = dict[VariableName];

    return std::make_pair(VariableValue, 1);


}

//ввод данных в список
bool Arithmetic::ListFormation(Node* nd, const std::string s,
std::ostream& out)
{
    out << "Next processed brackest value: " + s + "\n";
    size_t ind = 0, f;

    while (ind < s.size() && s[ind] == '(' || s[ind] == ' ') { ind++; }


    if ((s[ind] != '+' && s[ind + 1] == ' ') && s[ind] != '*' && (s[ind] !=
'-' && s[ind + 1] == ' '))
        return 0;

    Node* arg1 = new Node;
    nd->value = s[ind];

    ind++;
    while (ind < s.size() && s[ind] == ' ') { ind++; }

    if (s[ind] == '(')
    {
        f = ListFormation(arg1, ExtractBracketsValue(s, ind), out);
        if (!f)
```

```
	{
		delete arg1;
		return 0;
	}
}
else
{
	IntBoolPair result = ExtractValueForListFormation(s, ind);
	if (!result.second)
	{
		delete arg1;
		return 0;
	}
	arg1->value = result.first;
}

while (ind < s.size() && s[ind] == ' ') { ind++; }

if (s[ind] == ')')
{
	if (std::get<char>(nd->value) == '-')
	{
		nd->arguments = arg1;
		return 1;
	}
	else
	{
		delete arg1;
		return 0;
	}
}

Node* arg2 = new Node;

if (s[ind] == '(')
{
	f = ListFormation(arg2, ExtractBracketsValue(s, ind), out);
	if (!f)
	{
		delete arg1;
		delete arg2;
		return 0;
	}
}
else
{
	IntBoolPair result = ExtractValueForListFormation(s, ind);
	if (!result.second)
	{
		delete arg1;
		delete arg2;
		return 0;
	}

	arg2->value = result.first;
}
nd->arguments = std::make_pair(arg1, arg2);
```

```
    return 1;
}

//получение числа из строки для предыдущего метода
IntBoolPair Arithmetic::ExtractValueForListFormation(const std::string& s,
size_t& ind)
{
    std::string tmp = "";

    while (ind < s.size() && s[ind] != ' ' && s[ind] != ')')
    {
        tmp += s[ind];
        ind++;
    }
    if (InMap(tmp))
        return std::make_pair(dict[tmp], 1);

    std::istringstream stream(tmp);
    int result;
    stream >> result;
    if (stream.fail() || stream.peek() != EOF)
        return std::make_pair(0, 0);
    return std::make_pair(result, 1);
}

//получение значения хранящегося в очередных скобках для предпредыдущего
метода
std::string Arithmetic::ExtractBracketsValue(const std::string& s, size_t&
ind)
{
    int tmp_ind = ind, error = 0;
    std::string tmp_s = "";
    while (1)
    {
        tmp_s += s[tmp_ind];
        tmp_ind++;
        if (s[tmp_ind] == '(')
            error++;
        if (s[tmp_ind] == ')')
            error--;
        if (error < 0)
            break;
    }
    tmp_s += s[tmp_ind];
    ind = tmp_ind + 1;
    return tmp_s;
}

//вывод словаря
void Arithmetic::print_dict(std::ostream& out)
{
    for (auto i : dict)
        out << i.first << " = " << i.second << std::endl;
}

//распаковка значений переменных
bool Arithmetic::ExtractVariableValues(std::string value, std::ostream&
out)
```

```
{
   if (value == "()")
        return 1;

   size_t ind = value.find(' ');
   std::string VariableName(value.begin() + 2, value.begin() + ind);

   if (InMap(VariableName))
   {
        out << "Incorrect data! One variable with two values!";
        return 0;
   }
   ind++;

   std::string VariableValue;
   while (value[ind] != ')')
   {
        VariableValue += value[ind];
        ind++;
   }
   UpdateMap(VariableName, stoi(VariableValue));

   value.erase(value.begin() + 1, value.begin() + ind + 1);

   return ExtractVariableValues(value, out);
}

std::string InputDialog()
{
   std::string ivar;
   std::cout << "Choose input format:\n\t1)Console\n\n\t2) Read from file
(Default file is located along the path:"
        "TestInput.txt)\n\tIf you want to change file location, you have
to enter path as second argument\n\t"
        "Don't forget to change all \'\\\' to \'/\'" << std::endl;
   std::getline(std::cin, ivar);
   return ivar;
}

std::string OutputDialog()
{
   std::string ovar;
   std::cout << "Choose output format:\n\t1)Console\n\n\t2)File (Default
file is located along the path: TestOutput.txt)\n\t"
        "If you want to change file location, you have to enter path as
second argument\n\t"
        "Remember that debagging output will be saved with programm
result" << std::endl;
   std::getline(std::cin, ovar);
   return ovar;
}

std::string MakePath(std::string s, std::string const default_path)
{
   //Составление пути
   std::string path;

   //запись полученного пути в переменную path
```

```
    for (auto i = ++s.begin(); i != s.end(); i++)
        if (*i != ' ')
            path += *i;

    //проверка на то, что путь был введен
    if (path.empty())
        path = default_path;
    return path;
}

void CallAndCheck(StringPair data_, std::ostream& out)
{
    Arithmetic ar;
    out << "\nEntered string: " + data_.first + ", Entered variable list:
";
    if (data_.second == "()")
        out << "empty\n";
    else
        out << data_.second + "\n";

    //Регулярное выражение, создаваемое с целью проверки корректности
формата строки, сопоставляющей имена переменных и их значения
    std::regex regular("\\((\\([a-zA-Z0-9_]+ [0-9+-]+\\))+\\)");

    ar.SetStringValue(data_.first);

    //Проверка корректности расстановки скобок
    int CheckResult = ar.CheckBrackets();
    if (CheckResult)
    {
        out << ((CheckResult > 0) ? "Oops, you forgot about close
bracket(s)\n\n" : "Oops you forgot about open bracket(s)\n\n");
        return;
    }

    //Проверка на корректность полученного списка переменных/значений
    if (data_.second != "()" && !std::regex_match(data_.second.c_str(),
regular))
    {
        out << "Incorrect variable list!\n\n";
        return;
    }

    //Сопоставление именам переменных их значений
    if (!ar.ExtractVariableValues(data_.second, out))
        return;

    if (data_.second != "()")
    {
        out << "Variable list after unpacking:\n";
        ar.print_dict(out);
    }

    //Замена power на его значение, а заодно и проверка на корректность
записи power
    if (!ar.RemakePowerToInt())
    {
        out << "Incorrect power option!\n\n";
```

```
		return;
	}
	out << "String version after remake power to int: " +
ar.GetStringValue() + "\n";

	Node* nd = new Node;

	if (!ar.ListFormation(nd, ar.GetStringValue(), out))
	{
		out << "Incorrect expression!\n\n";
		return;
	}
	out << "Result: " << nd->evaluate() << "\n\n";
	out << "-----------------------------------------------------------------
--------------------\n";

	delete nd;

}

int main(int argc, char** argv)
{
	//s - строка для считывания данных из консоли/файла, VarValue - строка
для считывания значений переменных из консоли/файла
	StringPair data_, iovar;

	//Для удобного запуска тестов на другом пк
	if (argc > 1)
	{
		//Создание потока ввода и чтение данных из него
		std::string tmp(argv[1]);
		std::ifstream in("/" + tmp);

		if (!in.is_open())
		{
			std::cout << "Bad way!\n";
			return 0;
		}

		if (in.eof())
		{
			std::cout << "File is empty!\n";
			return 0;
		}

		std::vector<std::string> FileStrings;
		std::string tmp_s;
		while (std::getline(in, tmp_s)) { if (!tmp_s.empty())
FileStrings.push_back(tmp_s); }

		if (FileStrings.size() % 2)
		{
			std::cout << "String count cannot be odd\n";
			return 0;
		}

		int of = 0;
		std::ofstream outf;
```

```
        iovar.second = OutputDialog();
        switch ((iovar.second)[0])
        {
        case '1':
             of = 0;
        case '2':
             outf.open(MakePath(iovar.second, "TestOutput.txt"));
             if (!outf.is_open())
             {
                  std::cout << "File doesn't exist!";
                  return 0;
             }
             of = 1;
        }

        std::ostream& out = (of) ? outf : (std::cout);
        for (size_t i = 0; i < FileStrings.size() / 2; i++)
                  CallAndCheck(std::make_pair(FileStrings[i],
FileStrings[i + 1]), out);
        return 0;
   }
   else
   {
        iovar.first = InputDialog();
        iovar.second = OutputDialog();
   }

   bool of = 1;
   std::ofstream outf;

   switch ((iovar.second)[0])
   {
   case '1':
        of = 0;
        break;
   case '2':
        of = 1;
        outf.open(MakePath(iovar.second, "TestOutput.txt"));
        if (!outf.is_open())
        {
             std::cout << "File doesn't exist!";
             return 0;
        }
        break;
   }

   std::ostream& out = !of ? std::cout : outf;

   //Выбор формата ввода(если запуск произошел без дополнительных
аргументов)
   switch ((iovar.first)[0])
   {
   case '1':
        std::cout << "Enter string and variable values (input format: ((x1
c1)(x2 c2)...) x1, x2, ... - variables, c1, c2,... - their values),\n"
             "If your variable list is empty left this ()\n";
        std::getline(std::cin, data_.first);
```

```
            std::getline(std::cin, data_.second);

            CallAndCheck(data_, out);

            break;
        case '2':
        {

            std::string path = MakePath(iovar.first, "TestInput.txt");
            std::ifstream in(path);
            if (!in.is_open())
            {
                  std::cout << "File doesn't exist!\n";
                  return 0;
            }

            std::vector<std::string> FileStrings;
            std::string tmp_s;
            while (std::getline(in, tmp_s)) { if (!tmp_s.empty())
FileStrings.push_back(tmp_s); }

            if (FileStrings.size() % 2)
            {
                  std::cout << "String count cannot be odd\n";
                  return 0;
            }

            for (size_t i = 0; i < FileStrings.size(); i += 2)
                  CallAndCheck(std::make_pair(FileStrings[i], FileStrings[i +
1]), out);
        }
        }

        return 0;
}
```